Finite Automata vs Regular Expressions, Non-Regular Languages

CS 154
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory

and "guessing"
Regular Languages are closed under all of the following operations:

- **Union:** \( A \cup B = \{ w \mid w \in A \text{ or } w \in B \} \)
- **Intersection:** \( A \cap B = \{ w \mid w \in A \text{ and } w \in B \} \)
- **Complement:** \( \overline{A} = \{ w \in \Sigma^* \mid w \notin A \} \)
- **Reverse:** \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \} \)
- **Concatenation:** \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)
- **Star:** \( A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \} \)
Regular Expressions

Computation as simple, logical description

A totally different way of thinking about computation:

What is the complexity of describing the strings in the language?
Inductive Definition of Regexp

Let $\Sigma$ be an alphabet. We define the regular expressions over $\Sigma$ inductively:

For all $\sigma \in \Sigma$, $\sigma$ is a regexp

$\varepsilon$ is a regexp

$\emptyset$ is a regexp

If $R_1$ and $R_2$ are both regexps, then

$(R_1R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regexps
Precedence Order: *

\[ \text{Example: } R_1 \cdot R_2 + R_3 = ((R_1 \cdot R_2) + R_3) \]
Definition: Regexps Represent Languages

The regexp $\sigma \in \Sigma$ represents the language $\{\sigma\}$

The regexp $\varepsilon$ represents $\{\varepsilon\}$

The regexp $\emptyset$ represents $\emptyset$

If $R_1$ and $R_2$ are regular expressions representing $L_1$ and $L_2$ then:

$(R_1 R_2)$ represents $L_1 \cdot L_2$

$(R_1 + R_2)$ represents $L_1 \cup L_2$

$(R_1)^*$ represents $L_1^*$

Example: $(10 + 0^*1)$ represents $\{0^k1 \mid k \geq 0\} \cup \{10\}$
For every regexp $R$, define $L(R)$ to be the language that $R$ represents.

A string $w \in \Sigma^*$ is *accepted by* $R$ (or, $w$ *matches* $R$) if $w \in L(R)$.

Example: 01010 matches the regexp (01)*0.
Assume $\Sigma = \{0,1\}$

$\{ w \mid w \text{ has exactly a single 1} \}$

$0^*10^*$
Assume $\Sigma = \{0,1\}$

What language does the regexp $\emptyset^*$ represent?

$\{\varepsilon\}$
Assume \( \Sigma = \{0,1\} \)

\{ w \mid w \text{ has length } \geq 3 \text{ and its 3rd symbol is } 0 \} 

\((0+1)(0+1)0(0+1)^*\)
Assume $\Sigma = \{0,1\}$

\[ \{ w \mid \text{every odd position in } w \text{ is a } 1 \} \]

\[ (1(0 + 1))^{\ast}(1 + \varepsilon) \]
DFAs $\equiv$ NFAs $\equiv$ Regular Expressions!

L can be represented by some regexp

$\iff$ L is regular
L can be represented by some regexp
⇒ L is regular
Given any regexp $R$, we will construct an NFA $N$ s.t. $N$ accepts \textit{exactly} the strings accepted by $R$.

Proof by induction on the \textit{length} of the regexp $R$:

\textbf{Base Cases (R has length 1):}

- $R = \sigma$
  \[ \xrightarrow{\sigma} \]

- $R = \varepsilon$
  \[ \rightarrow \]

- $R = \emptyset$
  \[ \rightarrow \]
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

1. $R = R_1 + R_2$
2. $R = R_1 R_2$
3. $R = (R_1)^*$
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

- $R = R_1 + R_2$ By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$
- $R = R_1 R_2$ But $L(R) = L(R_1 + R_2) = L_1 \cup L_2$
- $R = (R_1)^*$ so $L(R)$ is regular, by the union theorem!
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

- $R = R_1 + R_2$  
  By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$

- $R = R_1 \cdot R_2$  
  But $L(R) = L(R_1 \cdot R_2) = L_1 \cdot L_2$

- $R = (R_1)^*$  
  so $L(R)$ is regular by the concatenation theorem
**Induction Step:** Suppose every regexp of length \( < k \) represents some regular language.

Consider a regexp \( R \) of length \( k > 1 \)

Three possibilities for \( R \):

\[
R = R_1 + R_2 \\
R = R_1 R_2 \\
R = (R_1)^* 
\]

By induction, \( R_1 \) and \( R_2 \) represent some regular languages, \( L_1 \) and \( L_2 \)

But \( L(R) = L(R_1^*) = L_1^* \)

so \( L(R) \) is regular, by the **star theorem**
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

1. $R = R_1 + R_2$ By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$
2. $R = R_1 R_2$ But $L(R) = L(R_1^*) = L_1^*$
3. $R = (R_1)^*$ so $L(R)$ is regular, by the *star theorem*

Therefore: If $L$ is represented by a regexp, then $L$ is regular
Give an NFA that accepts the language represented by \((1(0 + 1))^*\)

Regular expression: \((1(0+1))^*\)
Generalized NFAs (GNFA)

L can be represented by a regexp
\
\Leftrightarrow
\L\

L is a regular language

Idea: Transform an NFA for L into a regular expression by removing states and re-labeling the arcs with regular expressions

Rather than reading in just 0 or 1 letters from the string on a step, we can read in entire substrings
A GNFA is a 5-tuple $G = (Q, \Sigma, R, q_{\text{start}}, q_{\text{accept}})$

$Q, \Sigma$ are states and alphabet

$R : (Q-\{q_{\text{accept}}\}) \times (Q-\{q_{\text{start}}\}) \rightarrow \mathcal{R}$

is the transition function

$q_{\text{start}} \in Q$ is the start state

$q_{\text{accept}} \in Q$ is the (unique) accept state

$\mathcal{R} = \text{set of all regular expressions over } \Sigma$
A GNFA is a 5-tuple $G = (Q, \Sigma, R, q_{\text{start}}, q_{\text{accept}})$

Let $w \in \Sigma^*$ and let $G$ be a GNFA. $G$ accepts $w$ if $w$ can be written as $w = w_1 \cdots w_k$ where $w_i \in \Sigma^*$ and there is a sequence $r_0, r_1, \ldots, r_k \in Q$ such that

- $r_0 = q_{\text{start}}$
- $w_i$ matches $R(r_{i-1}, r_i)$ for all $i = 1, \ldots, k$, and
- $r_k = q_{\text{accept}}$

$L(G) = \text{set of all strings that } G \text{ accepts} = \text{“the language recognized by } G\text{”}$
This GNFA recognizes $L(a^*b(cb)^*a)$

Is $aaabcbcbcbaba$ accepted or rejected?

Is $bba$ accepted or rejected?

Is $bcba$ accepted or rejected?
Add unique start and accept states
While the machine has more than 2 states:

Pick an internal state, **rip it out and re-label the arrows with regexps**, to account for paths through the missing state.
While the machine has more than 2 states:

Pick an internal state, **rip it out and re-label the arrows with regexps**, to account for paths through the missing state
In general:

While the machine has more than 2 states:
While the machine has more than 2 states:

In general:

$$R(q_1, q_2) R(q_2, q_2)^* R(q_2, q_3) + R(q_1, q_3)$$
R(q_0, q_3) = (a*b)(a+b)^*
represents L(N)
$R(q_0, q_3) = (a*b)(a+b)^*$

represents L(N)
\[ R(q_0, q_3) = (a*b)(a+b)^* \]
represents \( L(N) \)
Formally: Given an DFA, add $q_{\text{start}}$ and $q_{\text{acc}}$ to create $G$

For all $q, q'$, define $R(q, q')$ to be $\sigma$ if $\delta(q, \sigma) = q'$, else $\emptyset$

**CONVERT($G$):** *(Takes a GNFA, outputs a regexp)*

If #states = 2  return $R(q_{\text{start}}, q_{\text{acc}})$

If #states > 2

select $q_{\text{rip}} \in Q$ different from $q_{\text{start}}$ and $q_{\text{acc}}$

define $Q' = Q - \{q_{\text{rip}}\}$

define $R'$ on $Q' - \{q_{\text{acc}}\} \times Q' - \{q_{\text{start}}\}$ as:

$$R'(q_i, q_j) = R(q_i, q_{\text{rip}}) R(q_{\text{rip}}, q_{\text{rip}})^* R(q_{\text{rip}}, q_j) + R(q_i, q_j)$$

return $\text{CONVERT}(G')$

**Claim:** $L(G') = L(G)$
Theorem: Let $R = \text{CONVERT}(G)$. Then $L(R) = L(G)$.

Proof by induction on $k$, the number of states in $G$

Base Case: $k = 2$ \hspace{1cm} \text{CONVERT outputs } R(q_{\text{start}}, q_{\text{acc}}) \checkmark

Inductive Step:

Assume theorem is true for $k-1$ state GNFAs

Let $G$ have $k$ states. Let $G'$ be the $k-1$ state GNFA obtained by ripping out a state.

We already claimed that $L(G) = L(G')$

$G'$ has $k-1$ states, so by induction,

$L(G') = L(\text{CONVERT}(G')) = L(R)$

Therefore $L(R) = L(G)$. \hspace{1cm} \text{QED}$
The automaton has two states, \( q_1 \) and \( q_2 \), with transitions labeled as follows:
- From \( q_1 \) to \( q_1 \) on input \( bb \)
- From \( q_1 \) to \( q_2 \) on input \( a + ba \)
- From \( q_2 \) to \( q_1 \) on input \( b \)
- From \( q_2 \) to itself on input \( \varepsilon \)
- From \( q_1 \) to itself on input \( \varepsilon \)
\[ bb + (a + ba)b^*a \]

\[ (bb + (a + ba)b^*a)^* (b + (a + ba)b^*) \]
Convert the NFA to a regular expression
Convert the NFA to a regular expression
Convert the NFA to a regular expression

\[ (a + b)b^*b \]
Convert the NFA to a regular expression

\[ ((a + b)b^*b(bb^*b)^a)^*(\epsilon + (a + b)b^*b(bb^*b)^*) \]
DEFINITION

DFAs    <->    NFAs

Regular Languages    <->    Regular Expressions
Some Languages Are Not Regular:

Limitations on DFAs
Regular or Not?

\[ C = \{ w \mid w \text{ has equal number of } 1s \text{ and } 0s \} \]

NOT REGULAR!

\[ D = \{ w \mid w \text{ has equal number of occurrences of } 01 \text{ and } 10 \} \]

REGULAR!
\{ w \mid w \text{ has equal number of occurrences of } 01 \text{ and } 10 \} \\
= \{ w \mid w = 1, w = 0, \text{ or } w = \varepsilon, \text{ or } w \text{ starts with a } 0 \text{ and ends with a } 0, \text{ or } w \text{ starts with a } 1 \text{ and ends with a } 1 \} \\
1 + 0 + \varepsilon + 0(0+1)^*0 + 1(0+1)^*1 \\

Claim: \\
A string \( w \) has equal occurrences of 01 and 10 \\
\iff \( w \) starts and ends with the same bit.
The Pumping Lemma: Structure in Regular Languages

Let $L$ be a regular language

Then there is a positive integer $P$ s.t.

for all strings $w \in L$ with $|w| \geq P$

there is a way to write $w = xyz$, where:

1. $|y| > 0$ (that is, $y \neq \varepsilon$)
2. $|xy| \leq P$
3. For all $i \geq 0$, $xy^iz \in L$

Why is it called the pumping lemma? The word $w$ gets *pumped* into longer and longer strings...
Proof: Let M be a DFA that recognizes L

Let P be the number of states in M

Let w be a string where \( w \in L \) and \( |w| \geq P \)

We show: \( w = xyz \)

1. \( |y| > 0 \)
2. \( |xy| \leq P \)
3. \( xy^iz \in L \) for all \( i \geq 0 \)

There must exist \( j \) and \( k \) such that \( 0 \leq j < k \leq P \), and \( q_j = q_k \)
Let’s prove that $B = \{0^n1^n \mid n \geq 0\}$ is not regular

By contradiction. Assume $B$ is regular.
Let $P$ be the number of states in a DFA for $B$.
Let $w = 0^P1^P$

If $B$ is regular, then there is a way to write $w$
as $w = xyz$, $|y| > 0$, $|xy| \leq P$, and
for all $i \geq 0$, $xy^iz$ is also in $B$

Claim: The string $y$ must be all zeroes.

Why? Because $|xy| \leq P$ and $w = xyz = 0^P1^P$

But then $xyyz$ has more 0s than 1s  Contradiction!
end